Alaska
Fisheries Science
Center

National Marine
Fisheries Service

U.S DEPARTMENT OF COMMERCE

AFSC PROCESSED REPORT 2013-01

RMark: An R Interface for Analysis of Capture-Recapture Data with MARK

March 2013

This document should be cited as follows:

Laake, J. L. 2013. RMark: An R Interface for analysis of capture-recapture data with MARK. AFSC Processed Rep. 2013-01, 25 p. Alaska Fish. Sci. Cent., NOAA, Natl. Mar. Fish. Serv., 7600 Sand Point Way NE, Seattle WA 98115.

RMark: AN R INTERFACE FOR ANALYSIS OF CAPTURE-RECAPTURE DATA WITH MARK

Jeffrey L. Laake

National Marine Mammal Laboratory

Alaska Fisheries Science Center

7600 Sand Point Way NE Seattle WA 98115

Email: jeff.laake@noaa.gov

March 2013

ABSTRACT

`RMark` provides a formula-based interface for analysis of capture-recapture data with the program `MARK`. `RMark` reduces the development time of models and removes the potential for errors introduced from manual creation of design matrices in `MARK`. By providing an interface to `MARK` in R, all of the tools in R can be accessed to extract and manipulate data, plot or simulate. It also provides the potential to include `MARK` analysis in a reproducible document with `Sweave` and LaTeX. I demonstrate `RMark` using the commonly analyzed European dipper data.

CONTENTS

INTRODUCTION

The most comprehensive software package for analysis of capture-recapture data is the program MARK (White and Burnham 1999). While it is unparalleled in the range of models and quality of the user documentation (http://www.phidot.org/software/mark/docs/book/), the interface for building models can be limiting for large data sets and complex models. While there is some capability for automatic model creation in MARK, most models are built manually with a graphical user interface to specify the parameter structures and design matrices. Manual model creation can be useful during the learning process but eventually it becomes a time-consuming and sometimes frustrating exercise. Also it can be a very real and unnecessary potential source of error in the analysis. Finally, for those that analyze data from on-going monitoring programmes, there is no way to extend the capture-history in MARK, which means all of the models must be re-created manually unless you pad the data for all future sampling occasions.

To make my own analyses more reliable and reproducible (Donoho 2010), I developed RMark, an R (R Core Development Team 2012) package that provides a formula based interface for MARK. RMark has been available since 2005 and is on the Contributed R Archive Network (CRAN) (http://cran.r-project.org). RMark contains functionality to build models for MARK from formulas, run the model with MARK, extract the output, and summarize and display the results with automatic labeling. RMark also has functions for model averaging, prediction, variance components, and exporting models back to the MARK interface. In addition, all of the tools in R are available which enable a completely scripted analysis from data to results and inclusion into a document with `Sweave` (Leisch 2002) and LaTeX to create a reproducible manuscript.

Here I provide an overview of the `RMark` package. For more detailed documentation, refer to the online documentation at `http://www.phidot.org/software/mark/rmark/` and the help within the `RMark` package.

BACKGROUND

`RMark` does not fit models to data. `MARK` does the model fitting. `RMark` creates an input text file, runs `MARK` to fit the model to the data and extracts the results from the `MARK` output file. Thus, to understand `RMark`, some background on `MARK` is necessary. `MARK` can currently analyze data for 140 model variations. In the `MARK` interface, selecting Help/Data Types will show all of the models and which are supported by `RMark`. Even though the model parameters and their structures vary, `MARK` and `RMark` use a standard approach to model construction. I will use the Cormack-Jolly-Seber (CJS) model as an example here to describe model construction. Extension to other models is straightforward by incorporating additional parameters.

Capture-recapture data are typically represented as an encounter history. For CJS models, the encounter history is a sequence of zeroes and ones, where a 1 codes for an encounter and a 0 means the animal was missed. Each position in the capture history represents a sampling occasion that occur at a sequence of times $t_1, t_2, ..., t_m$ for m occasions. For example, a sequence of 101 means the animal was initially encountered (or released) on the first occasion at t_1, not encountered on the second occasion at t_2, and encountered on the third and last occasion at t_3.

The parameters in CJS are apparent survival (ϕ) and capture probability (p).

These parameters can differ by time, age, cohort, grouping factor (categorical) variables such as sex and individual numeric covariates like weight. The parameter structure for the model is specified with a parameter index matrix (PIM) which are typically different across types of parameters. For example, with four sampling occasions, a time PIM structure will have three potential survival parameters, one for each time interval between occasions (Table 1). A constant PIM would have a single index for each parameter (Table 1). A model with a time PIM for survival and constant PIM for p has four parameters and can be represented by Phi(t)p(.) where a "." implies constant. Alternative PIMs can be specified but the all-different PIM (Table 1) used by `RMark` allows construction of any possible model.

Typically, a model with all-different parameters is not useful but restricted (subset) models can be constructed with a design matrix. For example, with an all-different PIM, the design matrix for Phi(t)p(.) is shown in Table 2. If X is the design matrix and β is the vector of parameters, the real parameters, ϕ and p, are computed with a default inverse logit link function:

$$\begin{bmatrix} \phi \\ p \end{bmatrix} = [1+exp(-X\beta)]^{-1}$$

For the example in Table 2, we would have $\beta = (\beta_1, \beta_2, \beta_3, \beta_4)$. Some example real parameter calculations are $\phi_2 = [1+exp(-\beta_1 - \beta_2)]^{-1}$ and $p_6 = [1+exp(-\beta_4)]^{-1}$ where p_6 represents the sixth parameter in an all-different PIM for p but it is actually index value 12 and the twelfth row in the design matrix. Other link functions are available and some more natural link functions like log are used to bound real parameters above 0 (e.g. abundance).

In R, the function model.matrix can be used to construct design matrices with a formula from data. In this case, the data are not the capture histories nor necessarily information about the animals. To use `model.matrix`, `RMark` creates 'design data' which are data attributed to the possible parameters. The concept of design data will be unfamiliar to anyone using `MARK` and it should not be confused with a design matrix which is created from the design data. "Design data" could have also been called "parameter data" because they are data associated with the various parameters in the model. The best way to describe design data is with an example.

For a CJS model with triangular PIMs, each row in the PIM represents a cohort (animals initially first caught on the occasion), columns represent times/occasions, and diagonals represent age (or time since marking) (Table 3). `RMark` creates design data cohort, age, and time as factor (categorical) variables with labels shown in (Table 3). Also, it creates numeric variables Cohort, Time and Age and each has an origin of 0 (e.g. Time = 0 for time = 1990 in ϕ design data). In `MARK`, factor variables for individual animals (e.g. sex) are handled as groups and each group has a separate PIM for each parameter. If there was a group variable of sex for the simple example, the design data for ϕ and p would each have 12 rows and an added field 'sex' that could have values 'F' for the first six rows and 'M' for the last six rows.

Using the design data in Table 3, the formula `~time` for ϕ and `~1` (intercept only) for p would create the complete design matrix in Table 2. `RMark` uses `model.matrix` with the design data and formula for each parameter and then combines them into a complete design matrix for all of the parameters. Typically, β parameters are not shared so the design matrices are offset and the gaps are filled with zeros as shown below.

Phi design matrix	**0**
0	**p design matrix**

In Table 2, the design matrix for ϕ is in rows 1 to 6 and columns 1 to 3 and the design matrix for p is in rows 7 to 12 in column 4. Rows 7 to 12 in columns 1 to 3 and rows 1 to 6 in column 4 are filled with zeros. Occasionally model parameters describe a similar quantity (p and c in closed capture models) and they will share columns in the design matrix. `RMark` copes with this by combining the design data for the parameters prior to building the design matrix.

In developing models with many parameters, groups and occasions, it became clear that using all-different PIM structures resulted in very large design matrices with long run times in `MARK` and some models would simply exceed the available memory. To avoid this problem, `RMark` simplifies the PIM structure after building the design matrix to use only as many indices as there are unique rows in the design matrix. With the example and the model Phi(˜1)p(˜1) (same as Phi(.)p(.)), there are only two unique rows in the design matrix even though there are 12 rows with the all-different PIMs. Prior to building the input file for `MARK`, with these formulas, the indices 1-6 would be set to 1 and 7-12 would be set to 2. The simplified PIMs and a design matrix with two rows and two columns would be passed in the file to `MARK`. The all-different indices are maintained in `RMark` to handle model averaging of models because each model could have a different simplified PIM structure but all models will have the same all-different PIM structures. Simplification is transparent except that the labeling of real parameters in the `MARK` output file can be confusing. However, real parameter labeling in the summary output and displays in `RMark` will be correct.

DIPPER DATA EXAMPLE

As an example, I use the European dipper (*Cinclus cinclus*) capture-recapture data analyzed by Lebreton et al. (1992). To demonstrate the use of individual covariates for ϕ, I added an imaginary static covariate “weight” (set to a random value between 1 and 10). Also, for modelling ϕ I use a flood covariate with a value of 1 for 1982 and 1983 and 0 otherwise.

The dipper data are contained in `RMark` and can be retrieved using `data(dipper)` after attaching the package (`library(RMark)`). For your own data, you can use the functions convert.inp to convert an existing `MARK` input file or `import.chdata` which is a convenience wrapper for `read.table`, or you can use any of the data input functions in R. The only restriction on the data frame is that it must contain a variable named `ch` which is a character string and if the capture history represents more than one animal, the field name for capture frequencies is named `freq`. The frequency is assumed to be 1 for each history if the variable `freq` does not exist. Any number or type of additional fields can be included. The only restrictions are that grouping variables (e.g. sex) must be factor variables, and individual (animal-specific) covariates (e.g. weight) must be numeric. The only other restriction is on the naming of time-varying individual covariates.

Once the data are in the R workspace, a model can be run with a single call to the function `mark`; however, that is only useful for demonstration purposes. It is better to follow a four step procedure:

1. process the data to identify the model and data attributes (`process.data`),

2. make and modify (if necessary) the design data (`make.design.data`),

3. write a function for the set of models to be fitted, and

4. run the function to fit the models.

Clearly, more steps will ensue to look at goodness of fit, model averaging and prediction, and plotting results. But, here I'll focus on the initial four steps. The documentation provides more details on other aspects such as setting initial values and fixing real parameters. I use a naming convention for objects in the steps but any valid object names can be used.

Step 1 is analogous to the first step with `MARK` in which you specify the type of capture-recapture model and the various descriptors like time intervals. Many of the items you specify on the `MARK` screen like number of occasions, group labels, and individual covariate names are set for you in `RMark`. But others like title, number of mixtures and time intervals are specified as arguments. The function process.data creates a list structure that contains the original dataframe and the various attributes that describe the data and how it should be analyzed. This list structure is called a processed dataframe and it is used for analysis because it has the attributes that describe how the data should be treated (e.g. CJS vs JS). For this example, we need to specify that we want to use CJS ("recaptures only") as the model structure, the time value for the first occasion (begin.time) is 1981 so we can interpret the time labeling more easily, and that we want to use the factor variable "sex" to define groups. These are all passed as arguments in the call to `process.data` and the result is stored in `dipper.proc` as shown below:

```
> dipper.proc=process.data(dipper, model="CJS",
+               groups="sex", begin.time=1981)
> names(dipper.proc)
```

```
 [1] "data"            "model"           "mixtures"          "freq"
 [5] "nocc"            "nocc.secondary"  "time.intervals"    "begin.time"
 [9] "age.unit"        "initial.ages"    "group.covariates"  "nstrata"
[13] "strata.labels"   "counts"          "reverse"
```

```
> dipper.proc$nocc
```

```
[1] 7
```

```
> dipper.proc$time.intervals
```

```
[1] 1 1 1 1 1 1
```

Values such as number of occasions (`nocc`) are computed from the data and others like time intervals have default values. The data are in `dipper.proc$data`. Some elements in the list are unused depending on the model. Little needs to be done with the processed dataframe because it is just used as a container to store the data and its attributes. If you modify the original dataframe, just remember to run it through `process.data` again because the unchanged data are stored in the processed dataframe and that is what is used in the analysis and not the original dataframe. You'll also need to reprocess the dataframe if you wish to change processing arguments like group structure.

Step 2 is to create the design data and modify it, if necessary. Below we create the design data for this dipper example and although there are arguments that can be set with the function, none are needed here. The function `make.design.data` is passed the name of the processed dataframe and the result is stored in `dipper.ddl` where the suffix is short for design data list because the result is a list of dataframes, one for each type of parameter in the model and a list of `pimtypes` for each parameter.

```
> dipper.ddl=make.design.data(dipper.proc)
```

For CJS, the dataframes are `Phi` and `p` and each dataframe contains 42 records: 21 for females and 21 for males. There are five factor variables, group, sex, cohort, age and time and there are three numeric variables Cohort, Time and Age which are continuous versions of the respective factor variables with the lower case letter. The group variable is a composite of the values of all factor variables used to define the groups but when there is a single factor variable as in this case it is redundant.

With these default design data you can specify a fairly rich set of models; however, you are not limited to the default design data. Additional variables can be added (e.g. effort or environmental conditions) and used in the formulas. For example, you can add fields that bin age, time or cohort to constrains parameters to be the same within each bin. Also, you can define additional variables such as a flood variable with a value 1 to specify the years in which there was a flood. Floods affected survival in 1982 and 1983, so a variable flood is assigned 1 for 1982 and 1983 and everywhere else it is 0.

```
> dipper.ddl$Phi$flood=ifelse(dipper.ddl$Phi$time%in%1982:1983,1,0)
```

Any group, cohort or time-specific covariate (e.g. effort, weather) can be included in the design data and these can be used in the formula to create the design matrix. Below are the design data for the first seven `Phi` parameters for Females and Males excluding the indices:

```
> dipper.ddl$Phi[c(1:7,22:28),-(1:2)]

    group cohort age time occ.cohort Cohort Age Time    sex flood
1  Female   1981   0 1981          1      0   0    0 Female     0
2  Female   1981   1 1982          1      0   1    1 Female     1
3  Female   1981   2 1983          1      0   2    2 Female     1
4  Female   1981   3 1984          1      0   3    3 Female     0
5  Female   1981   4 1985          1      0   4    4 Female     0
6  Female   1981   5 1986          1      0   5    5 Female     0
7  Female   1982   0 1982          2      1   0    1 Female     1
22   Male   1981   0 1981          1      0   0    0   Male     0
23   Male   1981   1 1982          1      0   1    1   Male     1
24   Male   1981   2 1983          1      0   2    2   Male     1
25   Male   1981   3 1984          1      0   3    3   Male     0
26   Male   1981   4 1985          1      0   4    4   Male     0
27   Male   1981   5 1986          1      0   5    5   Male     0
28   Male   1982   0 1982          2      1   0    1   Male     1
```

Before I jump to step 3, let's examine some model formula that only use fields in the design data and I'll demonstrate the use of `model.matrix` in `RMark`. Any formula can be used to create the design matrix using the fields in the design data. The formula `~1` is equivalent in `MARK` to the constant or "." model, `~time` is equivalent to the "t" model and `~Time` is equivalent to the "T" model with a linear trend over time. Consider a model with time varying survival. This portion of the design matrix can be constructed as follows (note: output restricted to first six rows):

```
> dm=model.matrix(~time,dipper.ddl$Phi)
> head(dm,6)

  (Intercept) time1982 time1983 time1984 time1985 time1986
1           1        0        0        0        0        0
2           1        1        0        0        0        0
3           1        0        1        0        0        0
4           1        0        0        1        0        0
5           1        0        0        0        1        0
6           1        0        0        0        0        1
```

Notice that it uses treatment contrasts and the first level of the factor variable time (1981) is the intercept. In contrast, the pre-defined models in `MARK` uses the SAS convention of specifying the last factor level as the intercept. You can get this same behavior by re-leveling the factor variable:

```
> dipper.ddl$Phi$time=relevel(dipper.ddl$Phi$time,"1986")
> dm=model.matrix(~time,dipper.ddl$Phi)
> head(dm,6)

  (Intercept) time1981 time1982 time1983 time1984 time1985
1           1        1        0        0        0        0
2           1        0        1        0        0        0
3           1        0        0        1        0        0
4           1        0        0        0        1        0
5           1        0        0        0        0        1
6           1        0        0        0        0        0
```

Here is an example that creates a design matrix for an additive model of sex and flood. I show the first four rows of the female parameters and the last four rows of the male parameters:

```
> dm=model.matrix(~sex+flood,dipper.ddl$Phi)
> dm[c(1:4,37:40),]

   (Intercept) sexMale flood
1            1       0     0
2            1       0     1
3            1       0     1
4            1       0     0
37           1       1     0
38           1       1     0
39           1       1     0
40           1       1     0

>
```

Interactions are specified with the * operator as in this example:

```
> dm=model.matrix(~sex*flood,dipper.ddl$Phi)
> dm[c(1:4,37:40),]

   (Intercept) sexMale flood sexMale:flood
1            1       0     0             0
2            1       0     1             0
3            1       0     1             0
4            1       0     0             0
37           1       1     0             0
38           1       1     0             0
39           1       1     0             0
40           1       1     0             0
```

Step 3 involves writing a function that provides a set of parameter specifications for each type of parameter (e.g. `Phi` and `p`) and the code to fit the models. There are three sections to the function:

1. Creating the different specifications for each parameter. A parameter specification is a list that gives a formula and other optional arguments like a link and fixed real parameters. Each parameter specification should be stored in an object with the naming convention of parameter.model_name. For example, it could be explanatory like `Phi.sex_plus_flood` or numerical like `Phi.1`. The important aspect is that it includes the name of the parameter followed by a period and then some additional text to differentiate between the various specifications.

2. Call the function `create.model.list` with the type of model (e.g. CJS) to create each combination of the parameter specifications.

3. Call `mark.wrapper` which in turn calls the `mark` function to fit a model for each combination of the parameter specifications.

Below is an example function called `dipper.analysis` which has no arguments (empty parentheses). It has four parameter specifications for `Phi` and `p`. It calls `create.model.list` with a model type of "CJS" so it knows the names of the parameters to search for the list of specifications. The final line of code is the call to `mark.wrapper` which has as its first argument `cml` which is the object created to contain the list of models. It also has arguments `data` and `ddl` to specify the processed data and the design data list. All arguments of function `mark` can be passed through `mark.wrapper`. Below, I set `output = FALSE` to avoid all of the output being displayed. Because the call to `mark.wrapper` is the last line in the function it is returned as the result when `dipper.analysis` is called.

```
> # Create function with parameter specifications to fit models
> dipper.analysis=function()
+ {
+    # Create specifications for Phi and p
+    Phi.1=list(formula=~time)
+    Phi.2=list(formula=~-1+time,link="sin")
+    Phi.3=list(formula=~sex+weight)
+    Phi.4=list(formula=~flood)
+    p.1=list(formula=~1)
+    p.2=list(formula=~time)
+    p.3=list(formula=~Time)
+    p.4=list(formula=~sex)
+    # Create a list of combinations of parameter specifications;
+    # the argument "CJS" tells it to look for CJS parameters
+    cml=create.model.list("CJS")
+    # Call mark.wrapper; the arguments are cml and then like with
+    # the mark function the processed data and ddl are specified,
+    # but they must be named data= and ddl= because they are passed
+    # through to the mark function; I've also set output=FALSE
+    # so it doesn't show a summary for the 16 models but it will
+    # show the model being fitted and any notations or errors.
+    mark.wrapper(cml,data=dipper.proc,ddl=dipper.ddl,output=FALSE)
+ }
```

Step 4 invokes the function to fit the models and store the results in an object (`dipper.results = dipper.analysis()`). The object `dipper.results` is a list with class "marklist" that `RMark` creates and understands. The list contains the mark model object for each fitted model and the model selection table which is `dipper.results$model.table` for this set of models (Table 4). Typing `dipper.results` also displays the model selection table. The models are listed in increasing order of AICc and the row number is the model number which can be used to extract the model . For example, if I wanted to get a summary of the best model which is model 13, I could use:

```
> summary(dipper.results[[13]])
```

The summary provides the type of fitted model, number of parameters, -2*log likelihood, AICc, a summary of the beta parameters with labels created by `model.matrix`, and then the real parameters shown in the PIM format for each parameter and each group. For CJS, the PIMs are triangular and viewing the parameters in this way makes it easy to identify the model structure. In the real parameter output (Table 5), it is easy to see the differences across the sexes and times and the pattern in survival due to the flood covariate in 1982 and 1983. The parameters for model 2 will match those in Table 11 of Lebreton et al. (1992).

Next, I will examine a model for survival with sex and weight to describe how individual covariates are handled. In the design matrix, individual covariates are handled by specifying the covariate name which is "weight" in this example:

```
> dipper.results[[11]]$design.matrix
                                 Phi:(Intercept) Phi:sexMale Phi:weight
Phi gFemale c1981 c1 a0 o1 t1981 "1"             "0"         "weight"
Phi gMale c1981 c1 a0 o1 t1981   "1"             "1"         "weight"
p gFemale c1981 c1 a1 o1 t1982   "0"             "0"         "0"
p gFemale c1981 c1 a2 o1 t1983   "0"             "0"         "0"
p gFemale c1981 c1 a3 o1 t1984   "0"             "0"         "0"
p gFemale c1981 c1 a4 o1 t1985   "0"             "0"         "0"
p gFemale c1981 c1 a5 o1 t1986   "0"             "0"         "0"
p gFemale c1981 c1 a6 o1 t1987   "0"             "0"         "0"
                                 p:(Intercept) p:Time
Phi gFemale c1981 c1 a0 o1 t1981 "0"           "0"
Phi gMale c1981 c1 a0 o1 t1981   "0"           "0"
p gFemale c1981 c1 a1 o1 t1982   "1"           "0"
p gFemale c1981 c1 a2 o1 t1983   "1"           "1"
p gFemale c1981 c1 a3 o1 t1984   "1"           "2"
p gFemale c1981 c1 a4 o1 t1985   "1"           "3"
p gFemale c1981 c1 a5 o1 t1986   "1"           "4"
p gFemale c1981 c1 a6 o1 t1987   "1"           "5"
```

`MARK` replaces the value for “weight” for each animal and computes the design matrix and parameters for each animal. Inclusion of covariate names cannot be handled with `model.matrix`. `RMark` creates a dummy variable with value of 1 for any individual covariate in the formula, uses `model.matrix` to create the design matrix and then replaces the values with the character string for the covariate name (e.g. “weight”) for the appropriate columns. This enables an individual covariate to be used with interactions with design data variables and other individual covariates. With individual covariates there is no single set of real parameter estimates. However, the function `covariate.predictions` can be used to compute the estimates for real parameters identified by indices. The following code computes a range of values for the covariate and demonstrates the use of the `PIMS` function to obtain the indices for a parameter. The non-simplified PIMS are the same for all of the models.

```
> # Compute min and max weights and
> # use it to construct range for plotting
> minmass=min(dipper$weight)
> maxmass=max(dipper$weight)
> mass.values=minmass+(0:30)*(maxmass-minmass)/30
> # Look at pim values to get index for female
> # and male for real parameters
> PIMS(dipper.results[[11]],"Phi",simplified=FALSE)

group = sexFemale
     1981 1982 1983 1984 1985 1986
1981    1    2    3    4    5    6
1982         7    8    9   10   11
1983             12   13   14   15
1984                  16   17   18
1985                       19   20
1986                            21
group = sexMale
     1981 1982 1983 1984 1985 1986
1981   22   23   24   25   26   27
1982        28   29   30   31   32
1983             33   34   35   36
1984                  37   38   39
1985                       40   41
1986                            42

>
```

There is no time variation in survival for model 11, so I can use parameter index 1 for females and 22 for males. A range of indices could be used if survival varied by time or cohort. Also, instead of specifying a single model, I could have specified the model list and model-averaged estimates would be computed. The following code computes the estimates and plots the relationship and confidence interval (Figure 1).

```
> # Set up graphics device with 2 plot areas
> pdf("weight_plots.pdf")
> par(mfrow=c(2,1))
> # Compute and plot survival values for females
> Phibymass=covariate.predictions(dipper.results[[11]],
+     data=data.frame(weight=mass.values), indices=c(1))
> # Plot predicted model averaged estimates by weight
> # with pointwise confidence intervals
> plot(Phibymass$estimates$covdata, Phibymass$estimates$estimate,
+    type="l",lwd=2,xlab="Mass(g)",ylab="Female Survival",
+    ylim=c(0,1),las=1)
> lines(Phibymass$estimates$covdata, Phibymass$estimates$lcl,lty=2)
> lines(Phibymass$estimates$covdata, Phibymass$estimates$ucl,lty=2)
> # Compute and plot survival values for males
> Phibymass=covariate.predictions(dipper.results[[11]],
+   data=data.frame(weight=mass.values),indices=c(22))
> plot(Phibymass$estimates$covdata, Phibymass$estimates$estimate,
+  type="l",lwd=2,xlab="Mass(g)",ylab="Male Survival",
+  ylim=c(0,1),las=1)
> lines(Phibymass$estimates$covdata, Phibymass$estimates$lcl,lty=2)
> lines(Phibymass$estimates$covdata, Phibymass$estimates$ucl,lty=2)
> dd=dev.off()
```

SUMMARY

Beyond the nicety and ease of providing a formula based interface to MARK, use of RMark enables all of the other tools in R to be used for data manipulation, plotting etc. With the various database packages in R like RODBC or mysql, and data manipulation packages like reshape2, data extraction, manipulation and construction of capture histories can be automated eliminating the need for temporary files. The results of the fitted model can be further manipulated (e.g. variance components) and displayed graphically. Using the tools in R, you can also easily conduct simulations to produce capture histories, compute estimates from MARK and iterate for simulation testing or bootstrapping.

The real benefit of RMark is realized with the use of Sweave and a LaTeX publishing system to produce manuscripts that are reproducible and internally documented. This manuscript was created with those tools. All of the tables, figures and quantities in the text are inserted directly rather than manually with the obvious potential for error. Changing the manuscript to accommodate changes in the data only requires recompiling the manuscript which re-runs the analysis. Computer-intensive analysis can be cached and re-run selectively. Exchanging documents and data with other researchers provides a useful educational tool. When a question arises about how a particular value was computed, the code will either be in the document or external package. RMark does not preclude making errors but it does make them less likely and more easily traceable.

ACKNOWLEDGMENTS

The findings and conclusions in the paper are those of the author and do not necessarily represent the views of the National Marine Fisheries Service, NOAA.

CITATIONS

Donoho, D. L. 2010. An invitation to reproducible computational research. Biostatistics 11:385-8.

Lebreton, J. D., K. P. Burnham, J. Clobert, and D. R. Anderson. 1992. Modeling survival and testing biological hypotheses using marked animals: a unified approach with case studies. Ecological Monographs 62:67-118.

Leisch, F. 2002. Sweave: dynamic generation of statistical reports using literate data analysis. In W. Hardle and B. Ronz, editors, Compstat 2002 - Proceedings in Computational Statistics, pages 575-580. Physica Verlag, Heidelberg.

R Core Development Team. 2012. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. URL http: //www.R-project.org/.

White, G. C. and K. P. Burnham. 1999. Program MARK: survival estimation from populations of marked animals. Bird Study 46:120-139.

Table 1. -- Examples of constant, time and all-different PIMs for a CJS model with four annual sampling occasions starting in 1990. The PIMs are triangular because animals first caught on occasion i can only be recaptured on occasions i+1 and beyond.

	Constant Recapture			Time Recapture			All-different Recapture		
Initial Capture	1991	1992	1993	1991	1992	1993	1991	1992	1993
1990	1	1	1	1	2	3	1	2	3
1991		1	1		2	3		4	5
1992			1			3			6

Table 2. -- Design matrix for a CJS Phi(t)p(.) model with four annual sampling occasions starting in 1990 and all-different PIMs. Indices 1-6 are used for ϕ and 7-12 for p. The index specifies the row in the design matrix and the columns are the parameters. This is the default treatment contrast used in R in which the first level of a factor is the intercept and the remaining are 'treatments'. Survival refers to an interval between occasions and is labeled based on the beginning time.

Index	ϕ: Intercept	ϕ:1991	ϕ:1992	p: Intercept
1	1	0	0	0
2	1	1	0	0
3	1	0	1	0
4	1	1	0	0
5	1	0	1	0
6	1	0	1	0
7	0	0	0	1
8	0	0	0	1
9	0	0	0	1
10	0	0	0	1
11	0	0	0	1
12	0	0	0	1

Table 3. -- Design data for a CJS model with four annual sampling occasions starting in 1990 and all-different PIMs and age representing number of years since initial marking. The index is the row number in the parameters design data. The indices into the design matrix (called the all-diff-index) are 1 to 12 and determined by the specified order of parameters in the model. The design data for ϕ and p are slightly different because survival refers to the interval between occasions and the times and ages are based on the time at the beginning of the interval (e.g. age 0 survival is from age 0 to age 1) and capture refers to an occasion and the times and ages at that occasion.

Parameter	Index	Cohort	Time	Age
ϕ	1	1990	1990	0
	2	1990	1991	1
	3	1990	1992	2
	4	1991	1991	0
	5	1991	1991	1
	6	1992	1992	0
p	1	1990	1991	1
	2	1990	1992	2
	3	1990	1993	3
	4	1991	1992	1
	5	1991	1993	2
	6	1992	1993	1

Table 4. -- Model selection table for CJS analysis of dipper data. Deviance values change between models because for models without individual covariates, it is the residual deviance and for models with individual covariates it is -2 log-likelihood. The argument use.lnl can be set to display -2 log-likelihood instead of deviance.

	model	npar	AICc	DeltaAICc	weight	Deviance
13	Phi(~flood)p(~1)	3.00	666.16	0.00	0.46	77.63
15	Phi(~flood)p(~Time)	4.00	667.40	1.24	0.25	76.83
16	Phi(~flood)p(~sex)	4.00	667.59	1.43	0.22	77.02
9	Phi(~sex + weight)p(~1)	4.00	673.80	7.64	0.01	665.71
11	Phi(~sex + weight)p(~Time)	5.00	673.81	7.65	0.01	663.67
1	Phi(~time)p(~1)	7.00	674.00	7.84	0.01	77.25
5	Phi(~-1 + time)p(~1)	7.00	674.00	7.84	0.01	77.25
14	Phi(~flood)p(~time)	8.00	674.03	7.87	0.01	75.21
3	Phi(~time)p(~Time)	8.00	674.71	8.55	0.01	75.89
7	Phi(~-1 + time)p(~Time)	8.00	674.71	8.55	0.01	75.89
12	Phi(~sex + weight)p(~sex)	5.00	675.33	9.18	0.00	665.19
4	Phi(~time)p(~sex)	8.00	675.50	9.34	0.00	76.68
8	Phi(~-1 + time)p(~sex)	8.00	675.50	9.34	0.00	76.68
2	Phi(~time)p(~time)	12.00	681.71	15.55	0.00	74.47
6	Phi(~-1 + time)p(~time)	12.00	681.71	15.55	0.00	74.47
10	Phi(~sex + weight)p(~time)	9.00	681.81	15.65	0.00	663.37

Table 5. -- Estimates of survival from best model for female dippers.

	1981	1982	1983	1984	1985	1986
1981	0.607	0.469	0.469	0.607	0.607	0.607
1982		0.469	0.469	0.607	0.607	0.607
1983			0.469	0.607	0.607	0.607
1984				0.607	0.607	0.607
1985					0.607	0.607
1986						0.607

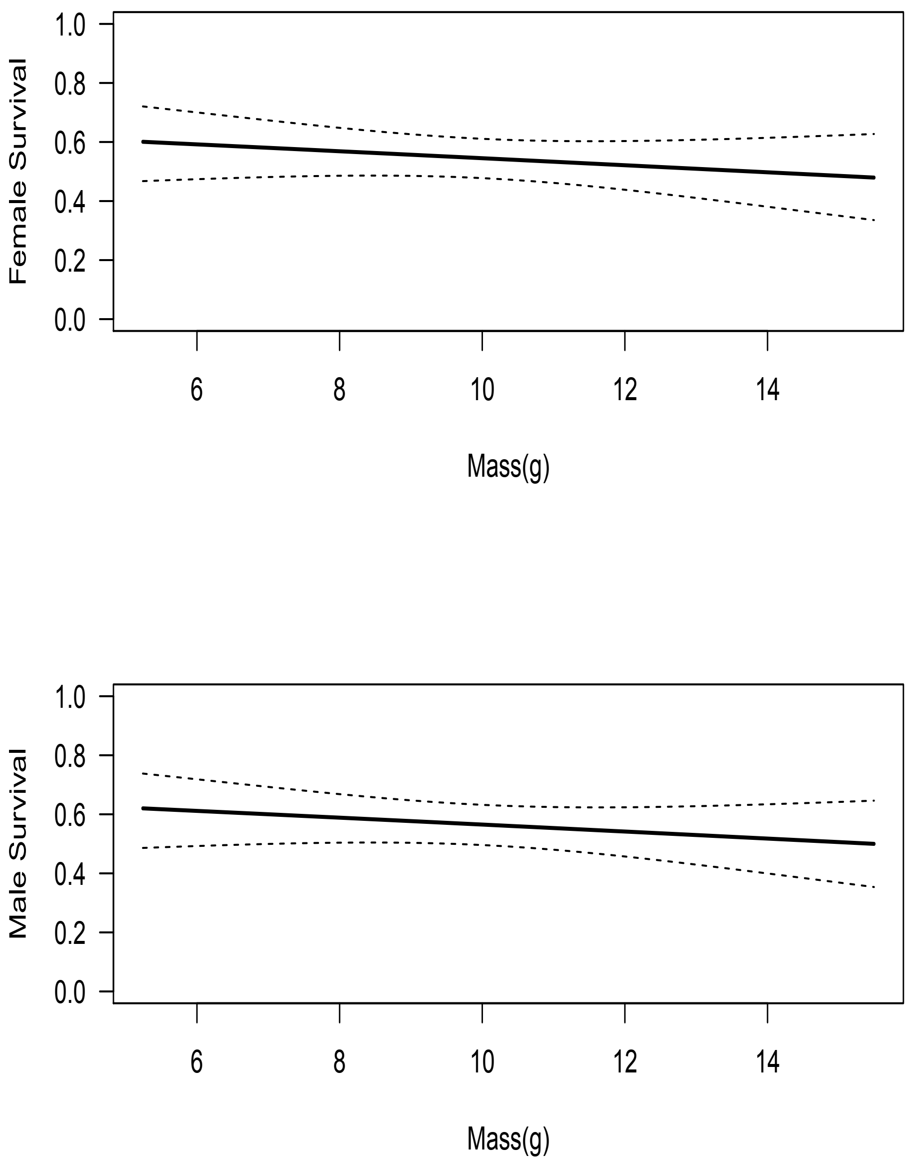

Figure 1. -- Predicted female and male dipper survival at various initial values of mass(g).